CAMP-BDI: A Pre-emptive Approach for Plan Execution Robustness in Multiagent Systems

CAMP-BDI: A Pre-emptive Approach for Plan Execution Robustness in Multiagent Systems

Alan White[1], Austin Tate[2], Michael Rovatsos[3]

Artificial Intelligence Applications Institute
Centre for Intelligent Systems and their Applications
School of Informatics, University of Edinburgh, UK
[1]a.g.white@sms.ed.ac.uk, [2]a.tate@ed.ac.uk, [3]mrovatso@inf.ed.ac.uk

Keywords: Multiagent Teamwork, Belief-Desire-Intention, Planning, Capability, Robustness

Abstract. Belief-Desire-Intention agents in realistic environments may face unpredictable exogenous changes threatening intended plans and debilitative failure effects that threaten reactive recovery. In this paper we present the CAMP-BDI (Capability Aware, Maintaining Plans) approach, where BDI agents utilize introspective reasoning to modify intended plans in avoidance of anticipated failure. We also describe an extension of this approach to the distributed case, using a decentralized process driven by structured messaging. Our results show significant improvements in goal achievement over a reactive failure recovery mechanism in a stochastic environment with debilitative failure effects, and suggest CAMP-BDI offers a valuable complementary approach towards agent robustness.

1 Introduction

The Belief-Desire-Intention (BDI) approach is widely applied towards intelligent agent behaviour, including within realistic domains such as emergency response. Realistic environments are stochastic and dynamic; exogenous change during execution can threaten the success of planned activities, risking both intention failure and debilitative consequences. Current BDI architectural implementations typically employ reactive approaches for failure mitigation, replanning or repairing plans after failure; *Jason* agents (Bordini and Hübner [2006]), for example, define recovery plans explicitly triggered by goal failure(s). This may risk additional costs associated with recovering from debilitated post-failure states – or even risk recovery being *impossible*. Continuous short-term planning helps handle uncertainty, but risks inadvertently stymieing long-term goals – such as if resource requirements are not identified and subsequently lost to contention.

We suggest agents embodied with capability knowledge can use introspective reasoning to proactively avoid plan failure. The CAMP-BDI (Capability Aware, Maintaining Plans) approach presented in this paper allows agents to modify intended plans when they are threatened by exogenous change – supporting use of long term planning whilst allowing response to unanticipated world states. We contribute the following components as part of our approach;

- An algorithm for anticipatory plan repair behaviour, henceforth referred to as *maintenance*
- Extension to the distributed hierarchical team case, using structured communication to drive individual adoption of responsibility for, and performance of, maintenance
- A supporting architecture, providing the capability, dependency, and obligation knowledge required to support introspective reasoning during maintenance
- A policy mechanism allowing runtime tailoring of maintenance behaviour

An experimental implementation of CAMP-BDI was evaluated against a reactive replanning system, using a logistics environment where unpredictable exogenous events could occur during intention execution. Our results were gathered over multiple experimental runs, for several probabilities of negative failure consequences. Our proactive approach was observed to hold a significant advantage in goal achievement over a reactive approach, and to offer greater efficiency (in terms of planning calls) at higher probabilities of negative failure consequences (i.e. where it was more likely failure led to a world state that increased difficulty of recovery planning).

2 Motivating Example

Our motivating example is a logistics domain, where goals require delivery of cargo to a set location in a stochastic, dynamic, continuous and non-deterministic environment. Uncertainty arises from agent health state, weather conditions (rainstorms may flood roads or cause landslips), or emergence of 'danger zones' (hostile insurgent activity at a given location). Failure risks negative consequences including vehicle damage (and, if already damaged, being rendered unusable), stranding off-road, or cargo destruction with possible toxic contamination rendering roads unusable. Figure 1 depicts a *Truck* agent travelling a planned route from location *A* to *M*, when road *F*→*M* is rendered unusable by flooding – threatening *Truck*'s intended activity, *move*(F, M).

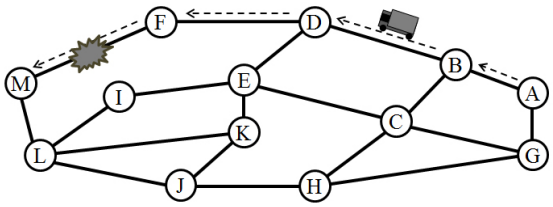


Fig. 1: Example of *Truck* executing a plan to travel from *A* to *M*.

We target environments where exogenous change causes divergence from the beliefs held at intention formation, failure risks debilitation, and resource contention prevents continual planning. This requires a *pre-emptive* approach; anticipating failure risks and proactively altering plans to compensate. Behaviour must extend to both local and multiagent contexts; i.e. if *Truck* cannot maintain sufficient confidence in meeting an obligation, the dependent agent should be able to compensate by modifying it's local (dependent) intended plan.

3 Architecture Components

In order to support a *pre-emptive* approach and allow introspective examination of intended plans, CAMP-BDI agents require the following meta-knowledge components. We suggest these can be regarded as subsets of agent *Beliefs*, although storage and retrieval semantics will be implementation-specific; the key element is that CAMP-BDI agents must specifically distinguish and consequently use this information in maintenance reasoning. Due to our approach of plan modification to avoid threatened failure, we adopt Simari and Parsons [2006]'s definition of an intention i as combining a goal and plan; i.e. $i = \{i_{goal}\ ,\ i_{plan}\}$.

3.1 Capabilities

Capabilities define meta-knowledge regarding activities performable, and goals achievable, by agents. Our capability model provides the information required for introspective maintenance

reasoning – it is also used to provide a representation of such information when conveyed within a dependency contract, allowing our algorithms to employ the same reasoning approach for both locally performed *and* delegated activities.

We define an activity a as similar to a *task* in a Hierarchical Task Network; where successful execution of a achieves some given state transition. A planned activity may represent either an atomic action or a (sub)goal, where performance of the latter entails execution of some subplan (i.e. an ordered sequence of activities). A capability $c(a)$, denoting the holding agents ability to perform a, has the following fields;

$$c(a) = <s,\ g(a),\ pre(a),\ eff(a),\ conf(a,\ B_a)>$$

- s: signature with name n and t parameters (s=$n(t_1,..., n_t)$); a specific capability instance c within the MAS can be uniquely identified by combining s and (identifier of) the agent holding $c(a)$.
- $g(a)$: defines an associated goal – a set of atoms ground to a. This can be used to distinguish between a defined purpose of a and its side-effects; i.e. *fly* and *drive* would achieve the same goal state (to arrive at some location), but with different total effects ($eff(a)$, below).
- *pre(a)*: preconditions (belief atoms) defining where a can be achieved - specifically, where use of $c(a)$ is not *guaranteed to fail*
- $eff(a)$: the complete set of post-effects of using $c(a)$ – i.e. $eff(a) = g(a) \cup$ side-effects($c(a)$).
- *conf*: $a \times B_a \rightarrow$ [0:1]; the *confidence* function; estimates the quality (in this context indicating likelihood of success) of using $c(a)$ to perform a, where belief set B_a gives the execution context. This supports identification of where exogenous change decreases optimality of a planned a – i.e. *conf*(*driveAlong*(*F*, *M*), B) is lower where $B \ni$ *slippery(F,M)* than $B \ni$ *dry(F,M)*.

In summary, holding $c(a)$ indicates that agent can achieve $g(a)$, with total post-effects $eff(a)$, provided $pre(a)$ holds in the execution context B_a, with the level of quality indicated by *conf*.

3.1.1 Capability typology

We define the *type* of a capability using two, overlapping categories;

Complexity: Primitive and Composite capabilities can be viewed as equivalent to basic and high level activities (Dhirendra *et al.* [2010]). *Primitive* capabilities represent atomic activities; *Composites* represent knowledge of one *or more* plans to perform some activity, or the ability to *form* a plan under specified preconditions. Each plan in an agent's plan library is represented by *exactly* one composite capability, meaning plans:capabilities have an n:1 relationship.

The activities found *within* plans will themselves correspond to capabilities. Composites can be seen as representing refinement options for (sub)goal activities within plans (i.e. requiring execution of some subplan), and also support continual planning by allowing reasoning whether plans exist for (as-yet unrefined) subgoals.

Locality: *Internal* capabilities represent activities an agent can perform itself; *External* capabilities represent those advertised by others, where a can be performed by delegation; as any decomposition is identified and performed by the resultant obligant(s) (i.e. the advertising agent), external capabilities are always primitive (e.g. are atomic from the *dependant*'s perspective).

3.1.2 The Confidence function

Certain state combinations may impact the likelihood of activity success, without being significant enough to include within preconditions (the *qualification problem* defined by McCarthy [1958]). The confidence function (*conf*(a, B_a)) is used to allow reasoning whether exogenous change has increased *risk* of failure, even where preconditions are not violated. A numerical value allows semantic-independent comparison between different internal and/or external capabilities for the same s, and supports varying levels of granularity (e.g.*yes=1*, *maybe=0.5*, *no=0*).

Estimation depends on both the capability type and a itself. If a is unground, confidence indicates the *general* ability of that agent to achieve g in B_a – an *abstract* estimation. If a is ground, additional semantic information can be used for *specific* estimation. *Primitive* capabilities will use a predefined calculation for both abstract and specific estimation – such as considering both past execution results (similar to Dhirendra *et al.* [2010]) and states in B_a. Implementation is domain and agent specific, requiring appropriate analysis of both.

External capabilities use a fixed, abstract confidence value as received in the relevant capability advertisement. Agents are unlikely to be able to share semantic knowledge required for *specific* estimation, as recipients may lack the modelling or sensory ability required to interpret. However, *specific* estimates are provided for delegated activities through the external capability field of contracts (see 3.2). We treat the actual advertisement process as implementation specific.

Composite capability confidence derives from the set of plans represented by that capability ($P_{capability}$). *Plan* confidence is the minimum held in a constituent activity (below, with *conf* expanded to consider a plan p as the first argument, where B_p is the execution context of the first activity in p); B_a requires updating with a_n's effects to estimate the execution context of a_{n+1}.

$$conf(p,\ B_p) = min_{a \in p} conf(a,\ B_a)$$

We assume the highest confidence plan is always selected for a goal. Composite capability confidence is taken as the highest of a selectable plan (where preconditions hold – if none are selectable, 0 is returned), where a_{goal} is the activity being performed using the composite capability and $B_{a_{goal}}$ it's execution context.

$$conf(a_{goal},\ B_{a_{goal}}) = \max_{\substack{p \in P_{capability} \\ pre(p) \subset B_{a_{goal}}}} conf(p,\ B_{a_{goal}})$$

This equates to formation and traversal of an AND-OR tree, similar to goal-plan trees described in Thangarajah *et al.* [2003], representing all potential plan and subplan execution 'paths' required to decompose and achieve a_{goal}. The return value derives from visiting every leaf activity ($O(n)$ worst case complexity, for n leaf nodes), originating from either a primitive or external capability confidence value. We assume cyclical loops cannot occur due to the decompositional nature of plans; this property is also required to prevent infinite loops in agent activity itself. Use of advertised confidence for external capabilities – rather than requesting potential dependants calculate a value locally – restricts semantic knowledge requirements to the advertising agent.

There is considerable scope for domain specific optimization of confidence calculation for both primitive and composite types. Average-case complexity can also be improved in contexts where a minimum threshold is being tested by using α-β pruning based on that threshold value. Finally, composite capabilities representing runtime planning abilities will require custom implementation of confidence estimation, similar to that used by primitive capabilities.

3.2 Obligation and Dependency Contracts

Our approach assumes dependency *contracts* are formed as early as possible, in advance of execution, to reserve agent capabilities and protect against possible agent resource contention. CAMP-BDI agents are aware of their obligation (to perform some activity upon request) *and* dependency (activities to be performed by some obligant) contracts. Contracts define mutual beliefs between dependants and obligants regarding a delegated activity – our algorithms require the following fields be represented and established during contract formation;

– The **activity** to be performed by the obligant(s) for the dependant.
– **Causal link** states; states that will be established by planned dependant, as effects of activities in the plan, prior to execution of the delegated activity.

– An **external capability**, used by obligant(s) to convey the (anticipated) post-effects and confidence for the activity – the latter estimating the execution context using the causal link states. If there is more than one obligant, the individual obligant capabilities will be merged;
  - *Confidence* is set as the *minimum* individual obligant confidence
  - *Preconditions* are formed as the conjunction of all obligant preconditions
  - *Effects* are set as the union of all obligant post-effects
– A **maintenance policy**, used to guide maintenance behaviour (see 3.3).

3.3 Maintenance Policies

A maintenance policy defines specific fields, applied to a defined set of agents and/or capabilities, where both field values and the applicable agent-capability set are modifiable during runtime;
– **Threshold**: the minimum confidence (quality) value for an activity; runtime modification of this value also allows compensation for over-sensitive confidence estimation
– **Priority**: guides relative prioritisation within maintenance behaviour, when multiple activities in an i_{plan} are identified as under threat

Maintenance policies are used to tailor maintenance behaviour; agent-capabilities associated with activities that have greater (probability or severity) failure consequences can be given lower thresholds and higher priorities. This assists balancing the additional computational costs of maintenance (lower threshold values act to increase the likelihood of an agent attempting to identify a confidence-raising modification of the i_{plan}) against the benefits of avoiding failure.

Contract maintenance policies merge dependant and obligant policies – these are matched respectively to the capability mapped to the dependent's i_{goal} (where i_{plan} contains the delegated activity) and to that associated with the obligant and delegated activity (obligant's i_{goal}). To restrict changes to a minimal subset of the overall distributed plan, the merged maintenance policy uses the most constrained field values (lowest threshold and highest priority values) to ensure obligants must have attempted maintenance before informing dependants of confidence changes.

4 The CAMP-BDI algorithm

The reasoning cycle (algorithm 1) of a CAMP-BDI agent extends Rao and Georgeff [1995]'s generic BDI algorithm with contract formation and maintenance steps (the former to support information requirements of the latter). Intentions are selected before the *maintain* function attempts to diagnose and correct threats to i_{plan}, the performance of which may result in subsequent modification. The maintain function is also called following receipt of *obligationMaintained* messages, which convey changes in how obligants will perform a delegated activity and also signifies they have performed any required (possible) local maintenance. The *formAndUpdateContracts* function forms new, and updates existing, dependency and obligation contracts; this executes after *maintain*, to account for plan modifications and/or inherited dependency contract changes.

The *maintain* function (Algorithm 2) first forms a priority-ordered list (agenda) of maintenance tasks – each representing a threatened activity. The agent iterates through this agenda, terminating when a maintenance task is successfully handled *or* the agenda emptied. The function *handleMaintenanceTask* attempts to modify i_{plan} to address the issue represented by a given maintenance task, returning true if successful (false if i_{plan} is unchanged).

Separating agenda formation and handling allows the former to prioritize amongst the complete set of threatened activities. Decoupling of agenda formation (i.e. threat diagnosis) and handling processes also facilitates investigation into alternate approaches for either.

In our motivating example (Fig. 1), *handleAgenda* would identify – using the associated capability's *pre* field – that flooding of $F \rightarrow M$ violates the preconditions of *move*(F, M), and insert

Algorithm 1: The CAMP-BDI reasoning cycle with changes from Rao and Georgeff [1995]'s generic algorithm highlighted as **bold** text, denoting maintenance activities and contract formation/updates.

```
initializeState();
while agent is alive do
    D ← optionGenerator(eventQueue, I, B);
    i ← deliberate(D, I, B);
    if i ≠ null & i not waiting on a dependency to complete then
        i ← updateIntentions(D, I, B);
        B_i ← estimated execution context of i;
        maintain(i, B_i);
        formAndUpdateContracts(i);
        execute();
    for each obligationMaintained message ∈ eventQueue do
        i_dependency ← the associated dependant intention;
        B_dependency ← estimated execution context of i_dependency;
        maintain(i_dependency, B_dependency);
        formAndUpdateContracts(i_dependency);
    for each obligation contract ∈ agent's Obligations do
        if i = ∅ then
            i_ob_goal ← activity defined in obligation;
            i_ob_plan ← cached plan for obligation (to achieve i_ob_goal);
            i_ob ← i_ob_goal, i_ob_plan;
            B_ob ← execution context estimated using (causal links in obligation ∪
            B);
            maintain(i_ob, B_ob);
        formAndUpdateContracts(i_ob);
    getNewExternalEvents();
    I ← dropSuccessfulAttitudes();
    I ← dropImpossibleAttitudes();
    I ← postIntentionStatus();
```

Algorithm 2: The *maintain* function

```
Data: i – An intention; a plan i_plan to meet some goal i_goal
      B_i – The estimated execution context of the first activity in i_plan
handled ← false;
agenda ← new empty Agenda;
agenda ← the agenda returned by formAgenda(i_goal, i_plan, B_i, agenda);
while ¬ handled & ¬ agenda.isEmpty() do
    handled ← handleMaintenanceTask(agenda.removeTop());
update Dependency contracts;
if i is an Obligation then
    update contract and send to the dependant in an obligationMaintained message;
```

a corresponding maintenance task into the agenda. A subsequent *handlingMaintenanceTask* call for that maintenance task would (attempt to) modify i_{plan} such that i_{goal} can be achieved, either by avoiding use of *move*(F,M) or (if capable) removing the flooded state of $F \rightarrow M$.

If there are multiple possible intentions ($I \neq \emptyset$), the agent only attempts to maintain the specific $i \in I$ that has been selected for execution. We view intention selection as goal driven behaviour, such that maintenance changes to improve an i_{plan} will not invalidate the original choice to *select* that i. This avoids the cost of maintaining *every* potential intention prior to selection – especially as maintenance of unselected intentions risks being rendered futile by subsequent agent activity. We terminate after the first successfully handled maintenance task, as modifications may invalidate other maintenance tasks in the agenda; this also provides a guaranteed termination point. An alternative is to iteratively diagnose and handle until either an empty agenda is formed or handling fails, but this is likely to result in significantly higher computational cost.

4.1 Maintenance tasks

Maintenance tasks are data structures which define a threatened activity, details of the threat and handling requirements. A maintenance task mt defines an activity a, task *type* (*preconditions* or *effects*), estimated execution context B_a for a, estimated confidence $conf_a$ of a given B_a, and the maintenance policy mp_a associated with a and used to set confidence thresholds;

$$mt = < a, \textit{type}, B_a, \textit{conf}_a, mp_a >$$

Capability knowledge facilitates introspective reasoning for maintenance task generation. Activities are mapped to – in precedence order – internal capabilities, contract-contained external capabilities, and finally advertised external capabilities; this assumes activities are only delegated where necessary and that agents adopt the least complex (fewest activities) approach for performing any activity. If an activity can be met by several external capabilities, that with highest general confidence is selected; mirroring the most likely criteria for obligant selection. Maintenance tasks are ordered in an agenda first by (mp_a defined) priority, then by precedence within the plan.

Preconditions maintenance task are generated where a's preconditions do not hold in B_a, but it is valuable to preserve a within the plan – either due to it fulfilling a goal state, or to avoid (the costs of) cancelling a pre-existing dependency contract for a. This type indicates maintenance should first attempt to restore precondition states before considering modifications to replace a.

Effects maintenance tasks arise where either preconditions do not hold and a does not require preservation, or $conf_a$ is under mp_a*.threshold* (a is of unacceptable quality and at risk of failure). This indicates a should be replaced by an activity sequence that will achieve the same post-execution effects as a.

4.2 Agenda Formation

Agenda formation (algorithm 3) employs recursion to support hierarchical plan structures (i.e. where composite activities are achieved through sub-plans). Each leaf activity (primitive activity *or* a composite which does not yet have an associated subplan, as may occur when employing continual planning) is iterated through in scheduled execution order. The *getCapability* function associates each activity with it's representative capability; this knowledge is used to identify threats and insert representative maintenance tasks into the agenda, with B_a finally being updated with activity effects (estimating the execution context for the subsequent activity). The *consolidate* function merges multiple maintenance tasks for the same subplan into a single maintenance task within the agenda, where appropriate. This consolidated maintenance task represents a need to maintain the entire subplan containing those activities – avoiding recurrent costs of re-diagnosing and handling each threatened activity individually, over multiple reasoning cycles.

Algorithm 3: The *formAgenda* function

Data: g – a goal met, or composite activity performed, by p
p – plan of *n* activities $\{a_0, a_1, ..., a_n\}$ to perform g
$agenda$ – priority ordered list of maintenance tasks; empty in initial (top-level) call
B_a – estimated execution context of a_0 in p
Result: $agenda$ updated with maintenance tasks for p
B_a updated with post-effects of p (used by recursion)
$B_{start} \leftarrow$ copy of B_a (for execution context estimation);
for *each activity* $a \in p$ **do**
  if a *is abstract* **then**
    return $agenda$, B_a;
  $c_a \leftarrow$ getCapability(a);
  if c_a = *null* **then**
    Add *effects* type maintenance task for a in B_a to $agenda$;
    Update B_a with effects of goal a;
  else if c_a *primitive* $\|$ *(*c_a *composite* $\&$ *(*a *is not decomposed into a subplan))* **then**
    if *maintenance task* mt *found for leaf activity* a **then**
      add mt to $agenda$;
      Update B_a with c_a.eff(a);
  else if c_a *composite* $\&$ *(*a *is decomposed into a subplan)* **then**
    $p_a \leftarrow$ subplan decomposing a;
    $agenda$, $B_a \leftarrow$ formAgenda(a, p_a, B_a, $agenda$);
  $agenda \leftarrow$ consolidate(g, $agenda$, B_{start});
return $agenda$, B_a;

4.3 Handling Maintenance Tasks

Handling a maintenance task (mt) requires modification of the i_{plan} containing $mt.a$, by identification and subsequent insertion of a new *maintenance plan* into i_{plan}. This behaviour is performed through the *handleMaintenanceTask* function (as called within the reasoning cycle given by Algorithm 1), which uses submethods *handlePreconditionsTask* (Algorithm 4) and *handleEffectsTask* (Algorithm 5) for the *preconditions* and *effects* types respectively. Capability knowledge is used to define an operator specification (reflecting currently accessible capabilities) and form the maintenance planning problem.

If a *preconditions* maintenance task cannot be handled, the algorithm generates and attempts to handle an equivalent *effects* maintenance task – relaxing the (preconditions maintenance) problem to allow replacement of $mt.a$ rather than fail from violated preconditions. For example, if *Truck* cannot restore preconditions for *move*(*F*,*M*) by unblocking road *F*→*M*, it will attempt to find an alternate method to achieve the required goal state *at*(*M*).

4.3.1 Performing Preconditions Maintenance

Preconditions maintenance (Algorithm 4) attempts to generate a plan re-establishing preconditions of $mt.a$, to be inserted prior to $mt.a$ (similar to *prefix* plan repair as defined by Komenda *et al.* [2014]). Generated maintenance plans are only inserted where their confidence is above $mt.mp_a$.*threshold*. This condition attempts to prevent subsequent maintenance of the i_{plan} arising from insertion of a suboptimal confidence plan, but is *not* applied if $mt.a$ is immediately due to execute – we deem any non-zero confidence plan preferable over guaranteed failure. In our mo-

tivating example (Fig. 1), successful preconditions maintenance would insert a (sub)plan which, when completed, removes the *flooded* state of road $F \rightarrow M$ before *move*(F, M) executes.

Algorithm 4: The *handlePreconditionsTask* function

Data: $task$ – a maintenance task
Result: **true** if a plan was found and inserted
$i_{mt} \leftarrow$ plan containing $task.a$;
$c_a \leftarrow$ getCapability($task.a$);
Define planning problem $prob_a$, with initial state = $task.B_{mt}$ and goal = c_a.pre($task.a$);
if *acceptable plan* $plan_a$ *solving* $prob_a$ *found* **then**
  Insert $plan_a$ into i_{mt} as predecessor of $task.a$, and **return** true;
return false;

4.3.2 Performing Effects Maintenance

Effects maintenance attempts to substitute a subset of the plan containing *mt.a*, with a new (sub)plan achieving identical effects. In our previous motivating example (Fig. 1), successful effects maintenance would substitute a new subplan for *move*(F, M) which achieves the associated i_{goal} – e.g. reforming the i_{plan} such that *Truck* (given a current location at D) will now travel through a (higher confidence offering) route $D \rightarrow E \rightarrow I \rightarrow L$.

Our algorithm (algorithm 5) adopts a similar approach to HTN plan repair – we use upwards recursion to re-refine composite activities (subgoals or the root i_{goal}), terminating when either an acceptable confidence (greater than $mt.mp_a.threshold$) maintenance plan is found and inserted, or the algorithm has reached the level of i_{goal} (i.e. attempted and failed to reform the entire i_{plan}). We trade-off the potential cost of multiple planning calls at goal/subgoal levels against the stability costs of complete replanning (Fox *et al.* [2006]).

The algorithm also considers potential costs from dependency cancellation, either from performing communication or the loss of external capability. Changes in circumstance after initial contract formation may render potential obligants subsequently unable to accept dependencies, even where now-cancelled dependency contracts previously existed – potentially stymieing maintenance planning if that external capability was necessary. To account for dependencies, we attempt two more restricted scope planning operations at the lowest level of iteration (the subplan containing $mt.a$). Firstly, if dependency contracts have been formed for $mt.a$ or it's successors in that subplan, the algorithm first attempts to generate a maintenance plan that directly and solely replaces $mt.a$; retaining successive activities and their associated dependency contracts. If any dependencies precede $mt.a$, the algorithm may also attempt *suffix* plan repair (Komenda *et al.* [2014]); where the generated maintenance plan replaces $mt.a$ and it's successors in that subplan, but preserves preceding activities. These two more constrained cases attempt to reduce disruption to a *distributed* plan performing team, at the cost of (potentially) requiring multiple planner calls.

The algorithm will, in the worst case, iterate and attempt to plan all levels of a hierarchical i_{mt}, including at the initial p_{mt} level twice (once for a failed preconditions maintenance task, and once for the replacement of $mt.a$ only), equivalent to $O((n+2)p)$ complexity (where n is the number of plan levels, and p the cost of planning). This, however, may still entail significant *actual* computational cost due to multiple planner invocations.

We are also investigating potential optimizations, including using policies to define conditions where planning is intractable (i.e. if *Truck* had been damaged to the extent *any* plan would have too low confidence) – allowing handling to terminate early and effectively delegate to any dependent. Another alternative is use of heterogeneous planners, allowing computationally re-

Algorithm 5: The *handleEffectsTask* function

Data: $task$ – a maintenance task
Result: **true** if a plan was found and inserted
$i_{mt} \leftarrow$ intended plan containing $task.a$;
if i_{mt} *is a hierarchical plan* **then**
  $p_{mt} \leftarrow$ subplan of i_{mt} containing a;
else
  $p_{mt} \leftarrow i_{mt}$;
$B_{mt} \leftarrow task.B_a$;
if a *not last in* $i_{mt} \parallel a$ *has subsequent dependencies* **then**
  $c_a \leftarrow$ getCapability(a);
  Define planning problem $prob_a$, with initial state = B_{mt} and goal = c_a.effects(a);
  if *acceptable plan* $plan_a$ *found for* $prob_a$ **then**
    Replace a in p_{mt} with $plan_a$;
    return true;
if a *not first in* $i_{mt} \parallel a$ *has preceding dependencies* **then**
  $a \leftarrow$ goal achieved by p_{mt};
  $c_a \leftarrow$ getCapability(a);
  Define $prob_a$, with initial state = B_{mt} and goal = c_a.effects(a);
  if *acceptable plan* $plan_a$ *found for* $prob_a$ **then**
    Replace p_{mt} from a inclusive with $plan_a$;
    return true;
while $a \neq$ *root goal of* i_{mt} **do**
  $a \leftarrow$ goal activity for p_{mt};
  $B_{mt} \leftarrow$ estimated execution context of a;
  $c_a \leftarrow$ getCapability(a);
  Define $prob_a$, with initial state = B_{mt} and goal = c_a.effects(a);
  if *acceptable plan* $plan_a$ *found for* $prob_a$ **then**
    Replace p_{mt} with $plan_a$;
    return true;
return false;

stricted agents to employ less flexible, but faster, approaches such as HTN planning or libraries. Finally, it is trivial to modify the effects maintenance algorithm to only perform top-level modification, if minimizing computational cost takes precedence over maximizing plan stability.

5 Distributed Behaviour

MASs use co-operative teams of agents to achieve goals unattainable by individuals; activity failure impacts other team members and threatens the success of distributed plans. In our approach, we assume hierarchical agent teams arise from delegation to, and decomposition into plans by, obligants. We define a decentralized approach as the distribution of knowledge and capability across agents often renders centralized approaches infeasible for realistic domains. We apply the previously defined individual maintenance algorithms to the distributed context, using structured communication to drive successive adoption of maintenance responsibility by individual agents at increasingly abstract levels of the decompositional team hierarchy (Fig. 2).

The supporting architecture is critical in supporting this behaviour; dependency and obligation contracts allow *specific* capability information to be provided for a delegated activity. Placing external capabilities within contracts makes this information available to dependants, whilst offsetting semantic *knowledge* requirements to the actual obligant(s). As both internal and external capabilities share the same representative model, this allows an agent's maintenance reasoning to regard delegated activities in the same manner as those (to be) performed locally.

If an agent maintains an i_{plan} where the i_{goal} meets an obligation, the (waiting) dependent agent is messaged after maintenance completes. Dependants are viewed as quiescent during execution of a delegated activity; this allows maintenance of a dependency to be triggered in response to obligants completing their local maintenance. Obligants will maintain intentions both when performing (as an intention) an obligation, or when not presently pursuing any intention (Algorithm 1) – in the latter case, such that idle agents will act to maintain mutual beliefs with their dependent regarding the *future* achievement (or otherwise) of that delegated activity.

Dependants adopt maintenance responsibility if and when their obligants are unable to maintain confidence in their subpart of a greater distributed plan; restricting changes in a distributed plan to the 'lowest' (most specific) agent level. We informally refer to this as 'percolation' of maintenance responsibility – in that responsibility gradually moves upwards in the team hierarchy, until an agent has maintained an intention and produced an outcome acceptable to both itself and any direct dependant. In our motivating example, if *Truck* is unable to reach M (despite local maintenance), the dependant may alter it's own i_{plan} to instead use a Helicopter (not hindered by flooding) as an obligant to achieve the delivery i_{goal}.

We can summarize the resultant behaviour in general terms (Fig. 2) as follows;

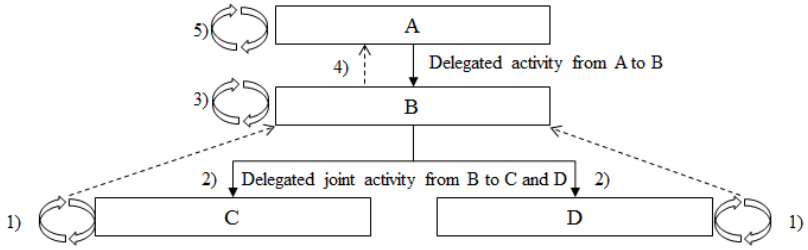


Fig. 2: The adoption of responsibility process in a hierarchical team, where *B* is an obligant of *A*, and *C* and *D* are obligants for a joint activity in *B*'s plan.

1. Agents *C* and *D* call *maintain* within local reasoning cycle(s).
2. *C* and *D* individually perform post-maintenance messaging; each sends a *obligationMaintained* message to *B* that includes contracts updated to account for any maintenance changes.
3. *B* calls it's *maintain* method upon receipt of *obligationMaintained* messages from all obligants. Information in the messaged, updated contracts is used to update the contract held by *B* for that dependency, which will itself be sent on to *A* after *maintain* completes.
4. *B* sends *A* post-maintenance messaging, again using *obligationMaintained* messages.
5. *A* calls *maintain* upon receipt of *B*'s post-maintenance message); as *A* is not an obligation, no further messaging is required.

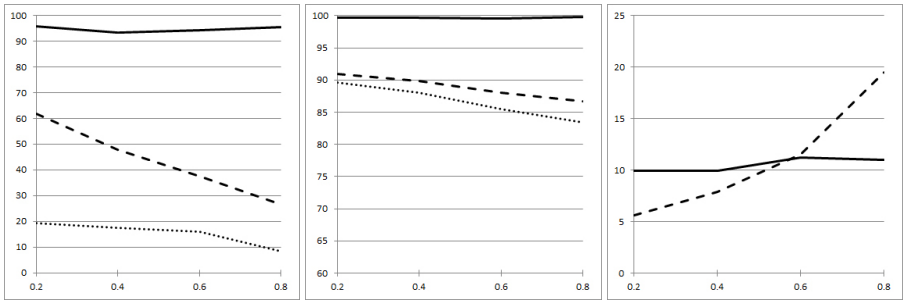
Contracts are employed to help synchronize this behaviour through defining a common maintenance policy for that activity, applied by both obligant(s) and dependant. As both share confidence threshold triggers, for a dependent to diagnose and attempt to handle an effects maintenance task the obligant must have first attempted the same. Although the above example indicates a linear approach to dependency formation, indirect 'self dependencies' can emerge – for example, in the above, *D* may form a dependency upon some other capability of *A* in the course

of performing it's own intention.

Our overall design aims to replicate HTN plan repair, but over a distributed plan; where each obligant's intended plan can be seen as analogous to an HTN task refinement. Agents assume responsibility for maintenance both when executing their own planned activities (as in Algorithm 1), and in response to an obligant maintaining it's own intention for an obligation. In the latter case, the dependant can use contractual information to judge whether that maintenance outcome is acceptable by it's *own* standards and modify the dependant i_{plan} if not.

6 Evaluation

CAMP-BDI was implemented by extending the *Jason* agent framework (Bordini and Hübner [2006]). We compared a MAS of CAMP-BDI agents against a system using reactive replanning, in our previously described motivating logistics domain. Three types of post-failure debilitation could occur, with defined probabilities for cargo damage, cargo spillage (requiring roads to be decontaminated), and for agent damage (with graded degrees and associated confidence loss). We evaluated performance under four probabilities: 0.2, 0.4, 0.6 and 0.8; representing 20, 40, 60 and 80% chance of an activity failure resulting in debilitation. These probabilities were applied individually for each debilitation type, albeit with cargo damage/spillage debilitation only possible if the failed activity was to load, unload or move whilst carrying cargo. Results for ten experimental runs, performed under fixed simulation seeds, were averaged and are presented in Fig. 3. A system with *no* failure mitigation was used as a worst-case indicator.



(a) y-axis; Average goals achieved (%)

(b) y-axis; Average activity success (%)

(c) y-axis; Average planner calls per goal achieved

Fig. 3: Experimental results; x-axis denotes post-failure damage probability. CAMP-BDI results are shown as solid lines, Replanning dashed, and Worst-Case dotted.

Our results show CAMP-BDI enjoyed significant advantage in goal success rate over replanning, increasing with the likelihood of debilitative failure consequences (Fig. 3a); CAMP-BDI maintained around 95% goal achievement for all consequence probability ranges, whilst replanning dropped from achieving 61.9% of goals at probability 0.2, to 26.6% at 0.8. In all three graphs CAMP-BDI shows fairly consistent performance; *avoidance* of failure (Fig. 3b) meant changes in *consequence* probability were unlikely to impact performance. In contrast, acceptance of failure as a 'trigger point' for reactive recovery meant replanning faced increasing difficulty in recovering (and achieving goals) from post-failure states as debilitation became more likely.

Worst case behaviour remained more variable (although always worst), as these agents would fail goals *immediately* upon activity failure regardless of debilitation (or otherwise) – unlike replanning, where agents would pursue the intended goal until all options were exhausted, potentially failing in multiple activities (with resultant accumulated debilitation) before recovery

attempts finally became futile. Goal and activity success rates of the *worst case* system were similarly variable; as no recovery mechanism was employed, each goal failure could be attributed to a single activity failure. This contrasts with reactive replanning, where agents would pursue an intended goal until all reactive replanning options were exhausted; each goal failure was potentially associated with multiple activity failures and consequent debilitations.

One obvious concern with a proactive approach is *cost*, particularly with CAMP-BDI's use of planning. Toyama and Hager [1997] note reactive approaches hold an advantage in only expending their costs following definitive, rather than potential, failure. Indeed, our results show CAMP-BDI performed significantly more planning calls at lower consequence probabilities (Fig. 3c); 9.91 calls per goal, compared to 5.62 for replanning at the lowest consequence probability. As the probability of post-failure debilitation increased, reactive replanning became significantly *less* efficient; an average 19.51 planning calls were required for each goal achieved at the highest consequence probability, compared to 11.03 for CAMP-BDI. This reflects an increasingly likelihood of debilitation stymieing *reactive* recovery – suggesting maintenance costs can be balanced against those incurred by failure. It may also still be preferable to employ a higher cost proactive approach, in domains where failure risks sufficiently severe consequences – such as if delivery goals are concerned with transport of nuclear waste or essential medical supplies.

7 Related Work

CAMP-BDI draws from a variety of existing work; our capability model captures the concepts of *know-how-to-perform*, *can-perform* and *know-how-to-achieve* defined by Morgenstern [1986]. Plan confidence estimation is similar to a subset of TÆMS quality metrics (Lesser *et al.* [2004]), such as *q_min*; future work may investigate alternate estimation approaches. He and Ioerger [2003] also suggest a quantitative estimation approach, in their case for producing maximally efficient schedules. Sabatucci *et al.* [2013] suggests use of capabilities (representing plans and viability conditions) to evaluate whether desires are achievable when selecting intentions.

Waters *et al.* [2014] suggest an intention selection mechanism prioritising the most constrained options, favouring those with least *coverage* (Thangarajah *et al.* [2012]), to increase the chance of *all* intentions completing. This differs from CAMP-BDI through considering arbitration between options in order to maximize intention throughput, rather than to ensure a specific intention succeeds. Whilst not explored in this paper, CAMP-BDI capability knowledge could facilitate similar reasoning during desire and intention selection. Plan execution monitoring approaches, such as SIPE (Wilkins [1983]), and plan repair approaches, such as O-Plan (Drabble *et al.* [1997]), share conceptual similarities with CAMP-BDI as both respond to divergence from expected states. CAMP-BDI differs through explicit focus on a multi-agent context, and use of confidence estimation to identify suboptimal activities.

Braubach *et al.* [2005] define two types of goals driving agent proactivity; those to *achieve* some state, and those to *maintain* it (over some defined period or under set conditions). Duff *et al.* [2006] further distinguishes reactive and proactive maintenance goals; the former requiring re-establishment of the state once violated, the latter constraining goal and plan adoption to prevent violation. In the reactive case, these drive adoption of achievement goals to re-establish violated (maintained) states; CAMP-BDI could consequently be used to maintain resultant intentions.

Precondition maintenance in CAMP-BDI can be viewed as similar in outcome to inferring proactive maintenance goals, corresponding to precondition states, and active until the relevant activity executes. Effects maintenance can be viewed as somewhat similar, in the sense that the loss of high-confidence associated states trigger plan modification; although our approach does not necessarily entail re-establishment of specific states if maintenance planning can identify an acceptable, alternate combination of activities. We also assume that plan formation mechanisms,

used both in intention formation and maintenance planning, will be implemented to recognise and respect any maintenance goals.

Work by Hindriks and Van Riemsdijk [2007] uses (limited) lookahead similar to CAMP-BDI, with regard to respecting proactive maintenance goals. They identify potential violations from adopted goals and plans using a goal-plan tree to anticipate future effects of adopted intentions. However, plans in this approach are treated as pre-defined and immutable; anticipated violation is suggested as best addressed by goal relaxation to allow alternative plan options. This may not be a viable approach in certain domains, if goals cannot be safely relaxed.

Duff *et al.* [2006] suggest a similar predictive approach, again using a goal-plan tree to filter goal adoption based upon effects of potentially usable plans. CAMP-BDI varies by more explicitly considering exogenous change, rather than potential goal/plan adoption, as a source of violation. Our approach also differs by focusing upon ensuring *existing* intentions avoid failure after exogenous change – proactive maintenance goals are typically employed more as constraints upon the *formation* and adoption of desires or intentions (although this encompasses adoption of subgoals and subplans in continual approaches).

Continual planning handles uncertainty by deferring planning decisions (desJardins *et al.* [2000]) – including decomposing certain abstract activities only upon execution. CAMP-BDI supports this approach, using composite capabilities to reason whether abstract activities can be met by (sub)plans. If planning incorporates sensing – representing knowledge requirements through preconditions and effects – these can be represented similarly within capabilities.

Markov Decision Processes (MDPs) are an alternate approach for acting within stochastic domains, using state transition probabilities and rewards to generate a *policy* defining the optimal activity in every possible state. Partially Observable MDPs (POMDPs) remove total knowledge assumptions through a probability map of state observations, used to infer actual states and define a solveable MDP. Whilst MDP approaches offer optimal behaviour, complexity issues render them intractable as state space increases. Schut *et al.* [2002] show BDI agents can handle relatively simple domains that are intractable for MDPs and approximate MDP performance (depending on time costs of runtime planning). Attempts to improve tractability typically involve abstraction – simplifying state spaces at the cost of optimality (Boutilier and Dearden [1994]).

Although BDI is viewed as a (time-efficient) alternative to MDP approaches, work has been performed to reconcile both; Simari and Parsons [2006] identify similarities and suggests possible mapping between policies and plans. Pereira *et al.* [2008] extend that work by defining an algorithm to form deterministic plans (for libraries) from POMDP policies – although this assumes the latter are formable offline. There is a risk that transition probability information is unavailable, or impractical to *learn* under domain time constraints. An MDP specification of a complex domain can also be non-intuitive, restricting practical usability; Meneguzzi *et al.* [2011] suggest a method to map more intelligible HTN domains onto MDPs – defining probabilities based upon state presence within operator preconditions, rather than probabilities in the *environment*. We defined CAMP-BDI under the assumption in realistic domains it is necessary to use deterministic plans, due to the above intractability and domain knowledge issues. A relationship can be envisaged between confidence estimation and MDP transition probabilities – although the former only requires a scalar *quality* estimate, rather than requiring exact probability estimation.

8 Conclusion

In this paper, we contribute CAMP-BDI – an approach towards distributed plan execution robustness using pre-emptive plan modification or *maintenance*. We have described the provision of capability knowledge, dependency contracts and dynamically modifiable policies to support pre-emptive maintenance through introspection, and depicted a structured messaging approach

for extending that individual agent behaviour to perform decentralized, distributed maintenance. Whilst we do not argue *all* failures can be prevented – that CAMP-BDI can replace reactive methods – we suggest it offers a valuable complementary approach. Finally, our supporting architecture may be useful for other robustness approaches or improving desire and intention selection; this helps justify the analytical costs involved in defining capability specifications.

CAMP-BDI does require gathering domain and agent information to model capability knowledge; consideration whether to employ our approach must balance the analytical and computational costs against the likelihood and severity of failure costs. We believe this domain analysis is a reasonable requirement, as such knowledge is used to form planning operators (or plan libraries) for agents. Even if fully granular and/or probabilistic confidence estimation is not possible, we view it as plausible for 'risky' world states to be identified and incorporated. Time-weighted success records can potentially be used for confidence estimation; similar to Dhirendra *et al.* [2010]'s use in learning plan execution contexts.

Our future work intends to focus upon maximizing gains from failure avoidance whilst minimizing planning costs; such as using policies to regulate maintenance behaviour, investigating potential specification of proactive maintenance goals within policies, and methods to focus 'computational expenditure' where avoiding failure is of greatest importance. Optimization of confidence estimation and agenda formation remains another aspect of interest, although we anticipate many of the most effective approaches will be domain specific.

Acknowledgements

This work was funded with support from EADS Innovation Works. Alan White would like to extend additional thanks to Dr. Stephen Potter for his invaluable help and advice. The authors and project partners are authorized to reproduce and distribute reprints and online copies for their purposes, notwithstanding any copyright annotation hereon.

References

R.H. Bordini and J.F. Hübner. BDI Agent Programming in AgentSpeak Using Jason. In F. Toni and P. Torroni, editors, *Computational Logic in Multi-Agent Systems*, volume 3900 of *Lecture Notes in Computer Science*, pages 143–164. Springer Berlin Heidelberg, 2006.

C. Boutilier and R. Dearden. Using Abstractions for Decision Theoretic Planning with Time Constraints. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 1016–1022. San Francisco, CA: Morgan Kaufmann, 1994.

L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal Representation for BDI Agent Systems. In R.H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*, volume 3346 of *Lecture Notes in Computer Science*, pages 44–65. Springer Berlin Heidelberg, 2005.

M.E. desJardins, E.H. Durfee, C.L. Ortiz Jnr., and M.J. Wolverton. A Survey of Research in Distributed, Continual Planning, 2000.

S. Dhirendra, S. Sebastian, P. Lin, and S. Airiau. Learning Context Conditions for BDI Plan Selection. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 325–332, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

B. Drabble, J. Dalton, and A. Tate. Repairing Plans On-the-fly. In *Proceedings of the NASA Workshop on Planning and Scheduling for Space*, 1997.

S. Duff, J. Harland, and J. Thangarajah. On Proactivity and Maintenance Goals. In *AAMAS-06*, pages 1033–1040, 2006.

M. Fox, A. Gerevini, D. Long, and I. Serina. Plan stability: Replanning versus plan repair. In *In Proc. ICAPS*, pages 212–221. AAAI Press, 2006.

L. He and T.R. Ioerger. A Quantitative Model of Capabilities in Multi-Agent Systems. In *Proceedings of the International Conference on Artificial Intelligence, IC-AI '03, June 23 - 26, 2003, Las Vegas, Nevada, USA, Volume 2*, pages 730–736, 2003.

K.V. Hindriks and M.B Van Riemsdijk. Satisfying maintenance goals. In *IN PROC. OF DALT'07*. Springer, 2007.

A. Komenda, P. Novák, and M. Pechoucek. Domain-independent multi-agent plan repair. *J. Network and Computer Applications*, 37:76–88, 2014.

V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. Nagendra Prasad, A. Raja, R. Vincent, P. Xuan, and X.Q. Zhang. Evolution of the GPGP/TÆMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):87–143, 2004.

J. McCarthy. Programs with Common Sense. In *Proceedings of the Teddington Conference on the Mechanisation of Thought Processes*, pages 77–84, 1958.

F. Meneguzzi, Y. Tang, K. Sycara, and S. Parsons. An approach to generate MDPs using HTN representations. In *Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities*, Barcelona, Spain, 2011.

L. Morgenstern. A First Order Theory of Planning, Knowledge, and Action. In *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, TARK '86, pages 99–114, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.

D.R. Pereira, L.V. Gonçalves, G.P. Dimuro, and A.C.R. Costa. Constructing BDI plans from optimal POMDP policies, with an application to AgentSpeak programming. In *Proc. of Conf. Latinoamerica de Informática, CLEI*, volume 8, pages 240–249, 2008.

A.S. Rao and M.P. Georgeff. BDI Agents: From Theory to Practice. In *In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, 1995.

L. Sabatucci, M. Cossentino, C. Lodato, S. Lopes, and V. Seidita. A Possible Approach for Implementing Self-Awareness in JASON. In *EUMAS'13*, pages 68–81, 2013.

M. Schut, M. Wooldridge, and S. Parsons. On Partially Observable MDPs and BDI Models. In *Selected Papers from the UKMAS Workshop on Foundations and Applications of Multi-Agent Systems*, pages 243–260, London, UK, UK, 2002. Springer-Verlag.

G.I. Simari and S. Parsons. On the Relationship Between MDPs and the BDI Architecture. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 1041–1048, New York, NY, USA, 2006. ACM.

J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and Avoiding Interference Between Goals in Intelligent Agents. In *IJCAI-03*, pages 721–726, 2003.

J. Thangarajah, S. Sardina, and L. Padgham. Measuring Plan Coverage and Overlap for Agent Reasoning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, pages 1049–1056, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

K. Toyama and G. Hager. If at First You Don't Succeed... In *Proc. AAAI*, pages 3–9, Providence, RI, 1997.

M. Waters, L. Padgham, and S. Sardina. Evaluating Coverage Based Intention Selection. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 957–964, Paris, France, May 2014. IFAAMAS. Nominated for Jodi Best Student Paper award.

D. E. Wilkins. Representation in a Domain-Independent Planner. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence. Karlsruhe, FRG, August 1983*, pages 733–740, 1983.